PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-245543
(43)Date of publication of application : 02.09.1992

(51)Int.Cl. G06F 9/42

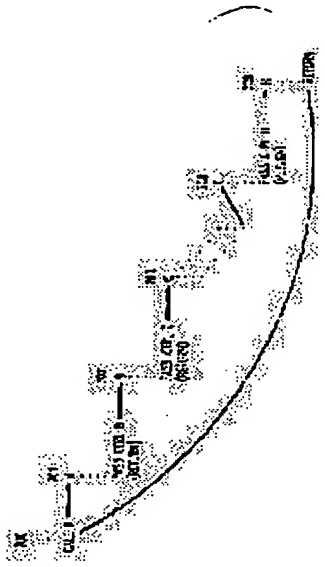
(21)Application number : 03-244386
(22)Date of filing : 29.08.1991

(71)Applicant : AMERICAN TELEPH & TELEGR CO <ATT>
(72)Inventor : DEBRULER DENNIS L

(30)Priority
Priority number : 90 577427 Priority date : 04.09.1990 Priority country : US

(54) METHOD AND DEVICE FOR EXECUTING SUBPROGRAM ACCESS AND COMPILER DEVICE

(57)Abstract:
PURPOSE: To attain a high speed access by stopping the execution of an accessing subprogram in accordance with each access in a subprogram sequence from the accessing subprogram during the execution of the subprogram and starting the execution of an accessed subprogram.
CONSTITUTION: A subprogram accessing sequence for plural different subprograms 200 to 209 starts a 1st access, executes at least one intermediate access and executes a final access to end the operation. During the execution of the subprograms 200 to 209, the execution of an accessing subprogram is stopped in accordance with each access in the sequence by the accessing subprogram and the execution of an accessed subprogram is started. Consequently small memory consumption and high speed access to the subprograms 200 to 209 can be executed.



LEGAL STATUS
[Date of request for examination]
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号

特開平4－245543

(43)公開日　平成４年(1992)９月２日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/42	３３０　Ａ	9189－5B		

審査請求　未請求　請求項の数57(全 21 頁)

(21)出願番号　特願平3－244386

(22)出願日　平成３年(1991)８月29日

(31)優先権主張番号　５７７４２７

(32)優先日　1990年９月４日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390035493
アメリカン　テレフオン　アンド　テレグラフ　カムパニー
AMERICAN TELEPHONE AND TELEGPAPH COMPANY
アメリカ合衆国．ニユーヨーク，ニユーヨーク，マデイソン　アヴエニユー　550

(72)発明者　デニス　エル．デブルラー
アメリカ合衆国　60515　イリノイ、ダウナズ　グローブ　メイン　ストリート　4720

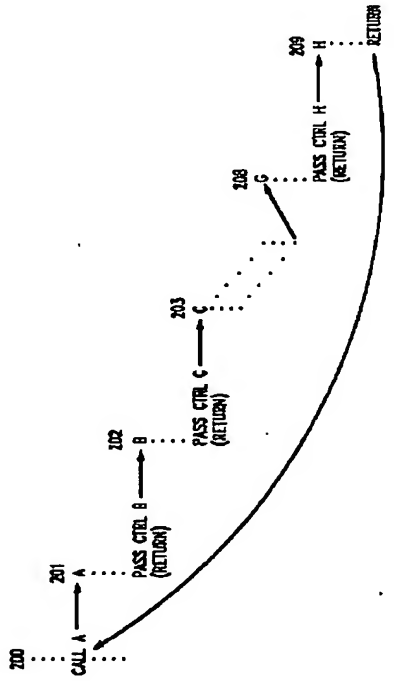
(74)代理人　弁理士　三俣　弘文

(54)【発明の名称】　サブプログラム呼び出しを実行する方法および装置とコンパイラ装置

(57)【要約】

【目的】　サブプログラムの呼び出し連鎖において、メモリ消費の少ない高速な呼び出しを可能にする。

【構成】　ＰＡＳＳ＿ＣＯＮＴＲＯＬ（図１１）と呼ばれる配置が、従来のＣＡＬＬおよびＲＥＴＵＲＮサブプログラム呼び出し系列（図２）にとってかわるものとして、従来のＲＥＴＵＲＮ文とともに使用される。本配置は、系列内の中間呼び出しを行ったサブプログラムへの中間的な復帰をすることなく、サブプログラム呼び出しの全系列から、系列を開始したサブプログラムへ直接復帰を実行する。本配置は、呼び出しの系列およびそこからの復帰を実行するために従来の実行スタック（１１４）を使用する（図１２～１４）。呼び出しの系列によって呼び出されたサブプログラムは実行スタック・フレーム（１６２０）を共有する。ＰＡＳＳ＿ＣＯＮＴＲＯＬの機能を実行するためのコンパイラ配置およびアプリケーション・プログラム実行配置が開示される。



1

【特許請求の範囲】

【請求項1】　複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法において、前記系列が、第1呼び出しで開始し、続いて少なくとも1個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記方法が、前記サブプログラム呼び出し系列内の呼び出しサブプログラムによるサブプログラムの各呼び出しに応じて、呼び出しサブプログラムの実行を停止し、呼び出されたサブプログラムの実行を開始するステップと、さらに前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納するステップと、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰元のサブプログラムの実行を停止し、前記実行スタックから前記命令ポインタを回復し、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記実行スタックから回復された命令ポインタによって指示される命令から、前記系列内の第1呼び出しを行ったサブプログラムの実行を再開するステップからなることを特徴とするサブプログラム呼び出しを実行する方法。

【請求項2】　前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項1の方法。

【請求項3】　前記実行を再開するステップが、前記実行スタックから回復された命令ポインタによって指示される命令の実行に分岐するステップからなることを特徴とする請求項2の方法。

【請求項4】　前記命令ポインタを回復するステップが、前記実行スタックから、前記格納された命令ポインタを削除するステップからなることを特徴とする請求項3の方法。

【請求項5】　前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記実行スタック上に格納するステップと、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項1の方法。

【請求項6】　前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記実行スタックから、実行コンテクストを、前記第1呼び出しを行ったサブプログラムの実行コンテクストに復元するステップからなることを特徴とする請求項5の方法。

【請求項7】　前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記実行スタック上に格納するステップと、前記系列内の中間呼び出しを行ったサブプログラム

2

の実行コンテクストに前記実行コンテクストを中間で復元することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記実行スタックから、実行コンテクストを、前記第1呼び出しを行ったサブプログラムの実行コンテクストに復元するステップからなることを特徴とする請求項1の方法。

【請求項8】　前記ポインタを格納するステップが、前記系列内の第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成するステップと、前記スタック・フレーム内に前記命令ポインタを格納するステップからなり、前記方法がさらに、前記系列内の第1呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームの作成を抑制するステップからなり、前記命令ポインタを回復するステップが、前記実行スタックから前記スタック・フレームを削除するステップを含むことを特徴とする請求項1の方法。

【請求項9】　前記系列内の第1呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項8の方法。

【請求項10】　前記系列内の第1呼び出しに応じて、前記第1呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記スタック・フレームに格納するステップと、前記系列内の前記第1呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項8の方法。

【請求項11】　前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記スタック・フレームから、実行コンテクストを、前記第1呼び出しを行ったサブプログラムの実行コンテクストに復元するステップからなることを特徴とする請求項10の方法。

【請求項12】　前記ポインタを格納するステップが、前記系列内の第1呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成するステップと、前記スタック・フレーム内に前記命令ポインタを格納するステップと、前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップからなり、前記方法がさらに、前記系列内の第1呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームを作成することなく前記呼び出されたサブプログラムの情報を前記スタックに格納するステップからなり、前記命令ポインタを回復するステップが、前記実行スタックから前記スタック・フレームをすべて削除するステップと、前記スタック・フレームに格納されたポインタによって指示される命令の実行に分岐するステップを含むことを特徴とする請求項1の方法。

【請求項１３】　前記系列内の第１呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制するステップからなることを特徴とする請求項１２の方法。

【請求項１４】　前記系列内の第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記スタック・フレームに格納するステップと、前記系列内の前記第１呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制するステップと、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記スタック・フレームから、実行コンテクストを、前記第１呼び出しを行ったサブプログラムの実行コンテクストに復元するステップからなることを特徴とする請求項１３の方法。

【請求項１５】　前記呼び出されたサブプログラムの情報を前記スタック上に格納する各ステップが、前記呼び出されたサブプログラムの情報のための領域を前記スタック・フレーム内に確保するステップと、前記呼び出されたサブプログラムのために確保された領域に前記呼び出されたサブプログラムの情報を格納するステップからなることを特徴とする請求項１４の方法。

【請求項１６】　前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップがそれぞれ、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納するステップからなることを特徴とする請求項１５の方法。

【請求項１７】　前記第１呼び出しに応じて前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップが、呼び出されたすべてのサブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保するステップからなり、前記呼び出されたサブプログラムの情報を前記スタック上に格納する各ステップが、前記呼び出されたサブプログラムの情報を前記１個の確保された領域に格納するステップをからなることを特徴とする請求項１４の方法。

【請求項１８】　前記呼び出されたサブプログラムの情報を前記スタック上に格納するステップがそれぞれ、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納するステップからなることを特徴とする請求項１７の方法。

【請求項１９】　複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法において、前記系列が、第１呼び出しで開始し、続いて少なくとも１個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記方法が、前記系列内の第１呼び出しに応じて、呼び出されるサブプログラムをＣＡＬＬするステップと（このステップは、前記第１呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納するステップを含む）、前記系列内の後続の各呼び出しに応じて、前記呼び出されたサブプログラムにＰＡＳＳ＿ＣＯＮＴＲＯＩするステップと（このステップは、実行が再開される命令へのポインタを格納することを抑制するステップを含む）、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記系列内の第１呼び出しを行ったサブプログラムの実行に、前記実行スタック上に格納された命令ポインタにおいて直接ＲＥＴＵＲＮするステップからなることを特徴とするサブプログラム呼び出しを実行する方法。

【請求項２０】　複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する装置において、前記系列が、第１呼び出しで開始し、続いて少なくとも１個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記装置が、複数の相異なるサブプログラムを実行する手段からなり、前記サブプログラム呼び出し系列内の呼び出しサブプログラムによるサブプログラムの各呼び出しに応じて、呼び出しサブプログラムの実行を停止し、呼び出されたサブプログラムの実行を開始する第１手段と、さらに前記系列内の第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する第２手段と、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰元のサブプログラムの実行を停止し、前記実行スタックから前記命令ポインタを回復し、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記実行スタックから回復された命令ポインタによって指示される命令から、前記系列内の第１呼び出しを行ったサブプログラムの実行を再開する第３手段からなることを特徴とするサブプログラム呼び出しを実行する装置。

【請求項２１】　前記系列内の第１呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制する手段からなることを特徴とする請求項２０の装置。

【請求項２２】　前記第３手段がさらに、前記実行スタックから回復された命令ポインタによって指示される命令の実行に分岐することによって前記復帰に対応することを特徴とする請求項2.1の装置。

【請求項２３】　前記第３手段がさらに、前記実行スタックから、前記格納された命令ポインタを削除することによって前記復帰に対応することを特徴とする請求項２２の装置。

【請求項２４】　前記第２手段がさらに、前記系列内の

5

第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記実行スタック上に格納し、前記サブプログラムの実行手段が、前記系列内の前記第１呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制することを特徴とする請求項２０の装置。

【請求項２５】　前記第３手段がさらに、前記実行スタックから、実行コンテクストを、前記第１呼び出しを行ったサブプログラムの実行コンテクストに復元することによって、前記復帰に対応することを特徴とする請求項２４の装置。

【請求項２６】　前記第２手段がさらに、第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記実行スタック上に格納し、前記第３手段がさらに、前記系列内の中間呼び出しを行ったサブプログラムの実行コンテクストに前記実行コンテクストを中間で復元することなく、前記復帰に応じて、前記実行スタックから、実行コンテクストを、前記第１呼び出しを行ったサブプログラムの実行コンテクストに復元することを特徴とする請求項２０の装置。

【請求項２７】　前記第２手段が、前記第１呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成する手段と、前記第１呼び出しに応じて、前記スタック・フレーム内に前記命令ポインタを格納する手段からなり、前記サブプログラムを実行する手段が、前記系列内の第１呼び出しに続く各呼び出しに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームの作成を抑制し、前記第３手段が、前記復帰に応じて、前記実行スタックから前記スタック・フレームを削除する手段を含むことを特徴とする請求項２０の装置。

【請求項２８】　前記サブプログラムを実行する手段が、前記系列内の第１呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制することを特徴とする請求項２７の装置。

【請求項２９】　前記第２手段がさらに、前記第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記スタック・フレームに格納する手段と、前記サブプログラムを実行する手段が、前記系列内の前記第１呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制することを特徴とする請求項２７の装置。

【請求項３０】　前記第３手段が、前記復帰に応じて、前記スタック・フレームから、実行コンテクストを、前記第１呼び出しを行ったサブプログラムの実行コンテクストに復元する手段を含むことを特徴とする請求項２９の装置。

【請求項３１】　前記第２手段が、前記第１呼び出しに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成する手段と、前記第１呼び出しに応じて、前記スタック・フレーム内に前記命令ポインタを格納する手段を含み、前記第１手段が、前記系列内の各呼び出しに応じて、前記呼び出されたサブプログラムの情報を前記スタックに格納する第４手段からなり、前記サブプログラムを実行する手段が、前記系列内の第１呼び出しに続く各呼び出しに応じて、呼び出されたサブプログラムのための新たなスタック・フレームを作成することを抑制し、前記第３手段が、前記復帰に応じて、前記実行スタックから前記スタック・フレームをすべて削除する手段と、前記復帰に応じて、前記スタック・フレームに格納されたポインタによって指示される命令の実行に分岐する手段を含むことを特徴とする請求項２０の装置。

【請求項３２】　前記サブプログラムを実行する手段が、前記系列内の第１呼び出しに続く各呼び出しに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納することを抑制することを特徴とする請求項３１の装置。

【請求項３３】　前記第２手段がさらに、前記第１呼び出しに応じて、前記第１呼び出しを行った呼び出しサブプログラムの実行コンテクストを前記スタック・フレームに格納する手段を含み、前記サブプログラムを実行する手段が、前記系列内の前記第１呼び出しに続く各呼び出しに応じて、コンテクストを前記実行スタック上に格納することを抑制し、前記第３手段が、前記復帰に応じて、前記スタック・フレームから、実行コンテクストを、前記第１呼び出しを行ったサブプログラムの実行コンテクストに復元する手段を含むことを特徴とする請求項３１の装置。

【請求項３４】　前記第４手段が、前記呼び出されたサブプログラムの情報のための領域を前記スタック・フレーム内に確保する手段と、前記呼び出されたサブプログラムのために確保された領域に前記呼び出されたサブプログラムの情報を格納する手段からなることを特徴とする請求項３３の装置。

【請求項３５】　前記第４手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納する手段からなることを特徴とする請求項３４の装置。

【請求項３６】　前記第４手段が、前記第１呼び出しに応じて、呼び出されたすべてのサブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保する手段と、前記系列内の各呼び出しに応じて、前記呼び出されたサブプログラムの情報を前記１個の確保された領域に格納する手段からなることを特徴とする請求項３３の装置。

【請求項３７】　前記第４手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前

記呼び出しサブプログラムによって前記呼び出されたサブプログラムに渡されたパラメータを格納する手段からなることを特徴とする請求項３６の装置。

【請求項３８】　複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する装置において、前記系列が、第１呼び出しで開始し、続いて少なくとも１個の中間呼び出しがあり、最終呼び出しで終了するものであり、前記装置が、前記系列内の第１呼び出しに応じて、呼び出されるサブプログラムをＣＡＬＬする第１手段と（この手段は、前記第１呼び出しを行った呼び出しサブプログラムの実行を再開する命令へのポインタを実行スタック上に格納することを含む）、前記系列内の後続の各呼び出しに応じて、前記呼び出されたサブプログラムにＰＡＳＳ＿ＣＯＮＴＲＯＬする第２手段と（この手段は、実行が再開される命令へのポインタを格納することを抑制することを含む）、前記系列内の中間呼び出しを行ったサブプログラムの実行を中間で再開することなく、前記系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、前記系列内の第１呼び出しを行ったサブプログラムの実行に、前記実行スタック上に格納された命令ポインタにおいて直接ＲＥＴＵＲＮする第３手段からなることを特徴とするサブプログラム呼び出しを実行する装置。

【請求項３９】　サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文から構成される複数の相異なるソース・サブプログラムからなるソース・プログラムを、オブジェクト命令から構成される複数のオブジェクト・サブプログラムからなるオブジェクト・プログラムへとコンパイルする手段からなるコンパイラ装置において、前記手段が、ソース・サブプログラムにおいてソース・サブプログラムを呼び出す文に遭遇することに応じて、前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する第１手段と、ソース・サブプログラムにおいてソース・サブプログラムを呼び出す文に遭遇することに応じて、前記文は複数の相異なるソース・サブプログラムの呼び出し文の系列内の第１呼び出し文であり、前記系列内の前記第１呼び出し文を除く文はそれぞれ前記系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムに含まれ、前記系列は前記第１呼び出し文で開始し、少なくとも１個の中間呼び出し文が続き、最終呼び出し文で終了するものであり、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する命令を生成する第２手段と、前記系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内の復帰文に遭遇することに応じて、前記

実行スタックから前記命令ポインタを回復し、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を中間で再開することなく、前記復帰文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記系列の第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、前記実行スタックから回復された命令ポインタによって指示される命令において、直接分岐する第３手段からなることを特徴とするコンパイラ装置。

【請求項４０】　前記ソース・プログラムをコンパイルする手段が、ソース・プログラムにおいて前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項３９の装置。

【請求項４１】　前記第３手段がさらに、前記復帰文に遭遇することに応じて、前記格納された命令ポインタを前記実行スタックから削除する命令を生成することを特徴とする請求項４０の装置。

【請求項４２】　前記第２手段がさらに、前記系列内の第１呼び出し文に遭遇することに応じて、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストを前記実行スタック上に格納する命令を生成し、前記ソース・プログラムをコンパイルする手段が、前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテクストを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項３９の装置。

【請求項４３】　前記第３手段がさらに、前記復帰文に遭遇することに応じて、前記実行スタックから、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストへ、実行コンテクストを復元する命令を生成することを特徴とする請求項４２の装置。

【請求項４４】　前記第２手段がさらに、前記第１呼び出し文に遭遇することに応じて、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストを前記実行スタック上に格納する命令を生成し、前記第３手段がさらに、前記復帰文に遭遇することに応じて、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストへ、前記実行コンテクストを中間で復元する命令を生成することなく、前記実行スタックから、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストへ、実行コンテクストを復元する命令を生成

9

することを特徴とする請求項３９の装置。

【請求項４５】　前記第２手段が、前記第１呼び出し文に遭遇することに応じて、前記実行スタック上に前記呼び出されたサブプログラムのスタック・フレームを作成する命令を生成する手段と、前記第１呼び出し文に遭遇することに応じて、前記スタック・フレーム内に前記命令ポインタを格納する命令を生成する手段からなり、前記ソース・プログラムをコンパイルする手段が、前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、前記呼び出されたサブプログラムのための新たなスタック・フレームを作成する命令の生成を抑制し、前記第３手段が、前記復帰文に遭遇することに応じて、前記実行スタックから前記スタック・フレームを削除する命令を生成する手段を含むことを特徴とする請求項３９の装置。

【請求項４６】　前記ソース・プログラムをコンパイルする手段が、前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項４５の装置。

【請求項４７】　前記第２手段がさらに、前記第１呼び出し文に遭遇することに応じて、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストを前記スタック・フレームに格納する命令を生成する手段と、前記ソース・プログラムをコンパイルする手段が、前記系列内の前記第１呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテクストを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項４５の装置。

【請求項４８】　前記第３手段が、前記復帰文に遭遇することに応じて、前記スタック・フレームから、実行コンテクストを、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストに復元する命令を生成する手段を含むことを特徴とする請求項４７の装置。

【請求項４９】　前記第２手段が、前記第１呼び出し文に遭遇することに応じて、前記実行スタック上に、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムのスタック・フレームを作成する命令を生成する手段と、前記第１呼び出し文に遭遇することに応じて、前記スタック・フレーム内に前記命令ポインタを格納する命令を生成する手段を含み、前記第１手段が、前記系列内の各呼び出し文に遭遇することに応じて、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの情報を前記スタックに格納する命令を生成する第４手段からなり、前記ソース・プログラムをコンパイルする手段が、前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、呼び出されたソース・

10

サブプログラムからコンパイルされたオブジェクト・サブプログラムのための新たなスタック・フレームを作成する命令の生成を抑制することを特徴とする請求項３９の装置。

【請求項５０】　前記ソース・プログラムをコンパイルする手段が、前記系列内の第１呼び出し文に続く各呼び出し文に遭遇することに応じて、実行が再開される命令へのポインタを前記実行スタック上に格納する命令の生成を抑制することを特徴とする請求項４９の装置。

【請求項５１】　前記第２手段がさらに、前記第１呼び出し文に遭遇することに応じて、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストを前記スタック・フレームに格納する命令を生成する手段を含み、前記ソース・プログラムをコンパイル手段が、前記系列内の前記第１呼び出し文に続く各呼び出し文に遭遇することに応じて、コンテクストを前記実行スタック上に格納する命令の生成を抑制し、前記第３手段が、前記復帰文に遭遇することに応じて、前記スタック・フレームから、実行コンテクストを、前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行コンテクストに復元する命令を生成する手段を含むことを特徴とする請求項４９の装置。

【請求項５２】　前記第４手段が、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの情報のための領域を前記スタック・フレーム内に確保する命令を生成する手段と、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムのために確保された領域に、前記呼び出されたサブプログラムからコンパイルされたオブジェクト・サブプログラムの情報を格納する命令を生成する手段からなることを特徴とする請求項５１の装置。

【請求項５３】　前記第４手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムによって、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムに渡されたパラメータを格納する命令を生成する手段からなることを特徴とする請求項５２の装置。

【請求項５４】　前記第４手段が、前記第１呼び出し文に遭遇することに応じて、呼び出し文によって呼び出されたソース・サブプログラムからコンパイルされたすべてのオブジェクト・サブプログラムの情報に対して単一の領域を前記スタック・フレーム上に確保する命令を生成する手段と、前記系列内の各呼び出し文に遭遇することに応じて、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの

情報を前記１個の確保された領域に格納する命令を生成する手段からなることを特徴とする請求項５１の装置。

【請求項５５】　前記第４手段が、前記実行スタック上のスタック・フレームに先行するパラメータ領域に、前記呼び出しソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムによって、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムに渡されたパラメータを格納する命令を生成する手段からなることを特徴とする請求項５４の装置。

【請求項５６】　前記コンパイル手段が、格納プログラムの制御下で動作する汎用データ・プロセッサを含み、前記格納プログラムは、前記データ・プロセッサが、前記ソース・プログラムを前記オブジェクト・プログラムへとコンパイルすることを可能にする命令のセットを含むことを特徴とする請求項３９の装置。

【請求項５７】　サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文から構成される複数の相異なるソース・サブプログラムからなるソース・プログラムを、オブジェクト命令から構成される複数のオブジェクト・サブプログラムからなるオブジェクト・プログラムへとコンパイルする装置が、ソース・サブプログラムにおいてソース・サブプログラムをＣＡＬＬする文に遭遇することに応じて、前記文は複数の相異なるサブプログラムの呼び出し文の系列内の第１呼び出し文であり、前記系列内の前記第１呼び出し文を除く文はそれぞれ前記系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムに含まれ、前記系列は前記第１呼び出し文で開始し、少なくとも１個の中間呼び出し文が続き、最終呼び出し文で終了するものであり、（ａ）前記第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタック上に格納する命令と、（ｂ）前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する第１手段と、ソース・サブプログラムにおいて前記系列内の後続の各呼び出し文に遭遇することに応じて、前記後続文がそれぞれソース・サブプログラムにＰＡＳＳ＿ＣＯＮＴＲＯＬし、（ａ）前記呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成し、（ｂ）実行が開始される命令へのポインタを格納する命令の生成を抑制する第２手段と、前記系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内のＲＥＴＵＲＮ文に遭遇することに応じて、前記実行スタックから前記

命令ポインタを回復し、前記系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を中間で再開することなく、前記ＲＥＴＵＲＮ文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、前記系列の第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、前記実行スタックから回復された命令ポインタによって指示される命令において、直接分岐する第３手段からなることを特徴とするコンパイラ装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、コンピュータにおけるサブプログラムの実行を連結するための装置に関する。

【０００２】

【従来の技術】サブプログラムは、他のプログラムの実行を呼び出し、呼び出されたプログラムから最終的にはその実行が復帰されるようなプログラムであるか、または、その実行が他のプログラムから呼び出され、それを呼び出したプログラムのような他のプログラムに実行を復帰することによって終了するようなプログラムである。もちろん、プログラムは、呼び出し側のサブプログラムであることも、呼び出される側のサブプログラムであることも可能である。サブプログラムは、ルーチン、サブルーチン、ファンクション、手続きのような、他の名前でも知られている。ここで使用される「サブプログラム」という語はこのようなすべての変化を包含する。

【０００３】サブプログラムは、構造化、またはモジュール化プログラミングにおいて広範に使用されている。このプログラミング法は、１個のタスクを一連のサブタスクに分解する、一般的に使用されているプログラミング技術である。構造化プログラミングはサブプログラムの長い連鎖を生じることがある。この場合、初期、またはメイン・サブプログラムがサブプログラムＡを呼び出し、続いてこれがサブプログラムＢを呼び出し、さらにこれがサブプログラムＣを呼び出す、などとなる。

【０００４】このようなサブプログラムの連鎖は、プロトコル・ハンドラ、トランザクション・ハンドラ、データベース照会サーバ、通話処理システムのようなある種のアプリケーション・プログラムに特徴的である。この種のアプリケーション・プログラムはまたスループット制約条件によっても特徴づけられる。他のすべてのことが等しい場合、スループットはプログラム実行速度に直接比例する。従って、サブプログラムの呼び出しおよび呼び出しからの復帰はできるだけ迅速に実行可能であることが重要である。

【０００５】もちろん、プログラム実行を高速化する１つの方法は、より高速なコンピュータを使用することである。しかし、コンピュータ速度は一般的にコンピュー

13

タ・コストに直接比例するため、この方法は高価である。さらに、技術開発は絶えず続いているので、技術は実現可能なプログラム実行速度に実際的な制限を設定せざるを得ない。従って、サブプログラムの呼び出しおよび呼び出しからの復帰は、与えられたコンピュータおよびコンピュータ技術でできるだけ高速なプログラム実行を達成するためには、できるだけ効率的に実現されることが重要である。

【０００６】サブプログラムの呼び出しおよびそのサブプログラムからの復帰を実現する従来広く行われている方法は、ＣＡＬＬおよびＲＥＴＵＲＮ文である。この高水準命令は、スタック（後入れ先出しメモリ構造）とともに作動する。実行を終了していない各サブプログラムはスタックに情報のフレームを有する。スタック・フレームには、引数すなわちパラメータ、局所変数、返値、およびサブプログラムに付随したその他の情報が格納されている。

【０００７】ＣＡＬＬ文によって、コール中のサブプログラムのコンテクストがスタック・フレームに格納され、コールされたサブプログラムにスタック・フレームが作成される。コンテクストは、汎用レジスタの内容、命令ポインタの内容、および条件コードのような情報を含む。ＲＥＴＵＲＮ文によって、リターンする、コールされていたサブプログラムのスタック・フレームがスタックから削除され、プロセッサは、リターン先の、コールしていたサブプログラムの格納コンテクストを復元する。

【０００８】メモリ内容およびプロセッサ状態のこのようなすべての操作には時間がかかるため、システム・スループットを減ずる。さらに、サブプログラム・コールの長い連鎖によって、スタックはメモリのかなりの領域を占有するため、他の用途に使用可能なメモリの量が減少する。このことは、明らかなメモリ制限の他に、メイン・メモリとのページのスワッピングのような活動の生じる頻度が増加することによってシステム・スループットを減ずることもある。

【０００９】従来のサブプログラムのコール・リターン装置のこれらの欠点のため、限定された範囲であっても、これらの欠点を改善する方法が必要である。

【００１０】

【発明が解決しようとする課題】１つの知られたアセンブラおよびリンク・エディタの組合せは、「ｌｅａｆｐｒｏｃ」として知られる機能を使用しており、これによって、リンク・エディタは標準的なコール命令を分岐リンク（ＢＡＬ）命令に変更する。ＢＡＬ命令は、次の（実行ではなくコンパイル順序に関する）命令のアドレスを、汎用レジスタのようなスタック外ロケーションに格納して、ＢＡＬ命令のオペランドによって指定された目的命令への通常の分岐動作を実行する。従ってスタックの状態は不変である。

14

【００１１】また、復帰は、ＢＡＬ命令が復帰アドレスを格納した汎用レジスタをオペランドとする分岐命令によって実現される。しかし、ＢＡＬ命令は、アセンブリ言語で書かれたサブプログラムの呼び出しにおける使用に制限され、しかもそのサブプログラムは、自分自身を含めて、他のサブプログラムを呼び出しておらず、数個より多くの汎用レジスタを必要としないものに制限される。

【００１２】ＵＮＩＸオペレーティング・システムは、「ｓｅｔｊｍｐ」および「ｌｏｎｇｊｍｐ」として知られる２個のペアの機能を使用している。一方のペアはオペレーティング・システム自体のアルゴリズムとして実現され、もう一方のペアはオペレーティング・システムへのユーザ・インタフェースにおけるライブラリ・ファンクションとして実現されている。

【００１３】オペレーティング・システム・カーネルのレベルでは、「ｓｅｔｊｍｐ」アルゴリズムはコンテクスト・スイッチを実行するが、セーブされたコンテクストをスタック上に格納する代わりに、それを新プロセスのメモリ領域（ｕ領域）にセーブする。このメモリ領域は、そのプロセスのコンテクストのみによってアクセスされる必要のあるプロセス制御情報を含む。従って実行は旧プロセスのコンテクストで継続する。

【００１４】カーネルがセーブしたコンテクストを再開したい場合は、「ｌｏｎｇｊｍｐ」アルゴリズムを使用し、これはｕ領域から、セーブされたコンテクストを復元する。この技術は、プロセスの概念を含むオペレーティング・システム・カーネルの実現に制限される。これは、オペレーティング・システム・カーネルにのみ使用可能なシステム・レベルの機能であり、アプリケーション・プログラマによる使用によってアクセス可能なものではない。

【００１５】ユーザ・インタフェースのレベルでは、「ｓｅｔｊｍｐ」ファンクションはコンテクスト・スイッチを実行し、旧コンテクストをスタックにセーブするが、そのコンテクストへのポインタは所定のスタック外ロケーションに格納する。その後、「ｌｏｎｇｊｍｐ」ファンクションは、パラメータとして所定のスタック外ロケーションのアドレスを与えられると、格納されたコンテクストに復帰し、復元される。

【００１６】「ｌｏｎｇｊｍｐ」ファンクションは従来のＲＥＴＵＲＮ文の多くのオーバヘッドを節約することが可能であるが、「ｌｏｎｇｊｍｐ」ファンクションは、ＣＰＵ命令サイクルやスタック・メモリ消費のような従来のＣＡＬＬ文のオーバヘッドを縮小することはない。しかも、ｌｏｎｇｊｍｐ命令を含むサブプログラムはすべて、ｓｅｔｊｍｐ命令を実行した親サブプログラム（サブプログラム呼び出しの連鎖における先行サブプログラム）を有していなければならない。

【００１７】従って、「ｌｏｎｇｊｍｐ」ファンクショ

15

ンを使用するサブプログラムは、任意のコンテクストから制限なしに呼び出されることが可能なサブプログラムとは異なり、「正常に」呼び出されることができない。さらに、ｓｅｔｊｍｐは、再帰的連鎖の親サブプログラムによって使用されることは可能だが、再帰的呼び出し連鎖（サブプログラム呼び出しの列であって、その列の最後の呼び出しが列の第１呼び出しを生じるサブプログラムを呼び出してループを形成するもの）の中で使用されることはできない。

【００１８】最後に、ある知られたインタプリタは、「限嗣再帰法」として知られる機能を利用している。自己参照再帰ファンクション内でのみ使用可能であり、再帰ファンクションが復帰前に実行する最後の作用が自分自身を呼び出す場合にのみ使用可能であるため、この機能は呼び出されたファンクションの反復のためのスタック・フレームの作成を抑え、ファンクションの呼び出し反復のフレームを再使用する。従来のＣＡＬＬおよびＲＥＴＵＲＮ文の列のオーバヘッドの多くを除去するために有効ではあるが、「限嗣再帰」機能は自己参照再帰ファンクション呼び出しのみに制限されるため、限定された適用可能性および有用性しか有しない。

【００１９】

【課題を解決するための手段】本発明は、従来技術の以上およびその他の欠点を解決することを目的とする。本発明によれば、例えば以下でＰＡＳＳ＿ＣＯＮＴＲＯＬと呼ばれる装置によって、相異なるサブプログラムの呼び出しの全系列からの、その系列を開始したサブプログラムへの直接の復帰が、その系列内の中間呼び出しを実行したサブプログラムへの復帰を妨げることなく実行される。

【００２０】本発明の第１目的によれば、複数の相異なるサブプログラムのサブプログラム呼び出しの系列を実行する方法および装置が開示される。この系列は第１呼び出しを開始し、続いて少なくとも１個の中間呼び出しを実行し、その後最終呼び出しで終了する。サブプログラムの実行中は、呼び出しサブプログラムによるサブプログラムの系列内の各呼び出しに応じて、呼び出しサブプログラムの実行は停止し、呼び出されたサブプログラムの実行が開始される。

【００２１】さらに、系列内の第１呼び出しに応じて、第１呼び出しを行った呼び出しサブプログラムの実行が行われる命令へのポインタが、実行スタックに格納される。例えば、対照的に、命令ポインタは、系列内のその他の呼び出しに応じては格納されない。従って、系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、復帰されたサブプログラムの実行は停止し、格納された命令ポインタが実行スタックから回復され、系列内の第１呼び出しを行ったサブプログラムの実行が、回復された命令ポインタによって指示された命令から再開される。最後に呼び出されたサブプログラムから、呼び出しの系列を開始したサブプログラムへのこの実行の分岐は直接であり、系列内の中間呼び出しを行うサブプログラムの実行の中間的な再開がない。

16

【００２２】最後に呼び出されたサブプログラムからの復帰が、呼び出し系列を開始したサブプログラムへ直接実行されるため、各呼び出しサブプログラムと呼び出されたサブプログラムの間の個々の復帰を実行し、個々の復帰を実行するための実行スタックを操作するために通常消費された時間が節約される。結果として、システム・スループットおよび性能はモジュール化実現において改善される。

【００２３】しかしながら、この方法および装置は、呼び出し系列およびそこからの復帰を実行するために、いくつかの従来の装置で行われているように例外的な特別のスタック外機構を使用するのではなく、従来の実行スタックを使用する。従って、従来技術とは異なり、サブプログラムはあらゆる状況で正常に呼び出されることが可能である。すなわち、１つのプログラム内で、同一のサブプログラムが、標準的なＣＡＬＬ装置およびＰＡＳＳ＿ＣＯＮＴＲＯＬ装置のいずれによって呼び出されることも可能であり、実行システムは、使用された呼び出し型を追跡する必要も、その呼び出し型の結果によって動作を調節する必要もない。

【００２４】系列内の呼び出しによって呼び出されるすべてのサブプログラムは実行コンテクストを共有することが望ましい。これによって、呼び出しおよび復帰の実行に消費される時間がさらに短縮される。系列内の第１呼び出しに応じて、第１呼び出しを実行した呼び出しサブプログラムの実行コンテクストが実行スタックに格納される。対照的に、実行コンテクストは、系列内のその他の呼び出しに対してはセーブされない。むしろ、存在するコンテクストが保持され共有される。また、系列内の最終呼び出しによって呼び出されたサブプログラムからの復帰に応じて、実行コンテクストはスタック上に格納されたコンテクストに復元される。もちろん、この復帰は、系列内の中間呼び出しを実行したサブプログラムのコンテクストの従来の中間的な復元なしに、直接的に実行される。

【００２５】さらに、呼び出し系列中に実行スタックによって消費されるメモリの量は、呼び出された各サブプログラムに対して新たなスタック・フレームを作成するのではなく、呼び出し系列によって呼び出されるすべてのサブプログラムが単一のスタック・フレームを共有することによって、大幅に縮小され限定されることが望ましい。系列内の第１呼び出しに応じて、呼び出されたサブプログラムに対するスタック・フレームが実行スタックに作成され、呼び出しサブプログラムの命令ポインタおよび実行コンテクストがそこに格納される。

【００２６】対照的に、系列内のその他の呼び出しに応じては、スタック・フレームは作成されず、命令ポイン

タおよび実行コンテクストもスタック・フレームに格納されない。呼び出された各サブプログラムに対する情報、例えば、局所および一時変数は、次の２つの方法のうちの１つによって共有スタック・フレームに格納される。１つの方法は、呼び出されたすべてのサブプログラムによって共有されるスタック・フレームの領域に格納する方法であり、もう１つの方法は、呼び出された各サブプログラムが拡張可能共有フレーム内に自分の使用のために確保した複数の領域のうちの１つに格納する方法である。

【００２７】パラメータは、各呼び出しサブプログラムによって、呼び出された各サブプログラムへ、汎用レジスタを通して、または、呼び出し系列を開始したサブプログラムのスタック・フレームの共有パラメータ領域を通して渡される。後者の共有パラメータ領域は実行スタックの共有スタック・フレームに先行する。また、最後に呼び出されたサブプログラムからの復帰に応じて、実行コンテクストおよび命令ポインタは現在のスタック・フレーム（この場合は共有スタック・フレーム）から復元され、現在の全スタック・フレームは、すべての復帰呼び出しによって行われるように、実行スタックから削除される。

【００２８】本発明の第２目的によれば、上記で特徴づけられる機能は、サブプログラム呼び出し文およびサブプログラム復帰文を含むソース・コード文からなる複数の相異なるソース・サブプログラムで構成されたソース・プログラムをコンパイルするための新しいコンパイラ装置によって実現される。特に、ソース・プログラムは、複数の相異なるソース・サブプログラムの呼び出しの文の系列を含む。

【００２９】系列内の第１呼び出し文を除いて、系列内の文は、系列内の先行する呼び出し文によって呼び出されたソース・サブプログラムにそれぞれ含まれる。この系列は、第１呼び出し文で開始し、少なくとも１個の中間呼び出し文が続き、最終呼び出し文で終了する。コンパイラ装置はソース・プログラムをコンパイルして、オブジェクト命令からなる複数のオブジェクト・サブプログラムで構成されるオブジェクト・プログラムを生成する。

【００３０】ソース・サブプログラム内で、ソース・サブプログラムを呼び出す文に遭遇すると、コンパイラ装置は呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、呼び出されたソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ分岐する命令を生成する。ソース・サブプログラム内で、系列内の第１呼び出しであるＣＡＬＬ文に遭遇すると、コンパイラ装置は、第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を再開する命令へのポインタを実行スタックに格納する命令を生成する。

【００３１】例えば、対照的に、コンパイラ装置は、系列内の後続呼び出しであるＰＡＳＳ＿ＣＯＮＴＲＯＬ文に遭遇した場合には、命令ポインタを格納する命令を生成しない。また、系列内の最終呼び出し文によって呼び出されたソース・サブプログラム内の復帰文に遭遇すると、コンパイラ装置は、実行スタックから格納された命令ポインタを回復し、復帰文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行から、系列内の第１呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行へ、実行スタックから回復された命令ポインタによって指示された命令において分岐し、系列内の中間呼び出し文を含むソース・サブプログラムからコンパイルされたオブジェクト・サブプログラムの実行を途中で再開することなく直接的に分岐する命令を生成する。コンパイラ装置はまた、上記で特徴づけられた付加的な機能を、対応するオブジェクト命令を生成することによって実現する。

【００３２】

【実施例】本発明を理解し評価する簡単な方法は、従来のコール・リターン装置と比較することである。従って、以下の説明はこの装置のかなり詳細な説明から開始される。

【００３３】図１は、格納プログラム制御下で動作する汎用データ・プロセッサの例（以下でコンピュータ１００と呼ぶ）のブロック図である。コンピュータ１００は、メモリ１０１および中央処理装置（ＣＰＵ）１０２を含む。ＣＰＵ１０２は通常、算術論理装置（ＡＬＵ）６０６およびレジスタ・ファイル６０５を含む。レジスタ・ファイル６０５は、少なくとも、命令ポインタ（ＩＰ）レジスタ６０３、スタック・ポインタ（ＳＰ）レジスタ６０２、フレーム・ポインタ（ＦＰ）レジスタ６０１、および複数の汎用レジスタ６０４を含む。

【００３４】ここでのメモリ１０１は、ＲＡＭ、ＲＯＭやディスクのような任意の形式の記憶装置を表す。これは、ＣＰＵ１０２によって実行されるプログラム１１１、および、その実行中にプログラム１１１によって使用される入力情報１１０を含む。ＣＰＵ１０２によるプログラム１１１の実行に従い、メモリ１０１はまた、入力情報１１０を使用してプログラム１１１の実行によって生成された出力情報１１２を含む。従って、ＣＰＵ１０２は式ｆ（ａ）＝ｂによって定義されるファンクション要素１１３を実現するとみなされることが可能である。ただし、ｆは実行プログラム１１１によって実現されるファンクション、ａは入力情報１１０、ｂは出力情報１１２である。

【００３５】プログラム１１１がコンピュータ１００によって実行されるためには、プログラムはコンピュータ１００の論理（例えばＡＬＵ６０６）によって理解され

19

る形式でなければならない。この形式は一般的にオブジェクト・プログラムと呼ばれている。しかし、ほとんどのプログラムは最初にオブジェクト形式で書かれることはなく、高水準のソース・プログラム形式で書かれる。

【００３６】従って、プログラムは、コンピュータ１００によって実行される前に、ソースからオブジェクト形式に変換されなければならない。この変換は、コンパイラ・オブジェクト・プログラム（または略してコンパイラ）によって実行される。従って、最初に、実行プログラム１１１はコンパイラであり、入力情報１１０はアプリケーション・ソース・プログラムであり、出力情報１１２は、コンパイラによってアプリケーション・ソース・プログラムから生成されたアプリケーション・オブジェクト・プログラムである。その後、アプリケーション・オブジェクト・プログラムが実行プログラム１１１となり、アプリケーション入力情報１１０上で動作して、アプリケーション出力情報１１２を生成する。

【００３７】実行中、プログラム１１１は、メモリ１０１内に実現された実行スタック１１４を使用して、少なくともいくつかの部分的な入力および出力情報を、他の一時的情報とともに一時的に格納する。図８を見れば、スタック１１４とは後入れ先出しメモリ構造である。これには次の２個のポインタが付随する。スタック・ポインタ（ＳＰ）６０２はスタック１１４上で次の空き記憶ロケーションを指示し、フレーム・ポインタ（ＦＰ）６０１は実行中のサブプログラムに付随したスタック部分の開始点すなわち最初の記憶ロケーションを指示する。このスタック部分はスタック・ポインタとフレーム・ポインタの間の範囲にあり、フレーム（例えばフレーム６１０、６１１、６１２）と呼ばれる。

【００３８】本出願の焦点は、コンパイラによって生成されたオブジェクト・プログラムによるスタック１１４の効率的な使用法と、コンパイラがオブジェクト・プログラムによるこの効率的な使用を引き起こす方法である。

【００３９】アプリケーション・ソース・プログラム１１０は、図２に示されたようなサブルーチン終端コールおよびリターンの連鎖を含むことがある。各サブプログラム２００～２０９はその中に他のコールを含んでもよい。このコールは連鎖の一部とは見なされず、本出願の焦点の主題ではない。本出願は、呼び出されたサブプログラムの終端コール、すなわち、呼び出しサブプログラムへのリターンの前に呼び出されたサブプログラムによって実行される最後の文のみに関する。

【００４０】図２に示されるように、この連鎖はメイン・サブプログラム２００によるＡサブプログラム２０１へのコールで始まる。Ａサブプログラム２０１は次にＢサブプログラム２０２をコールし、ただちにリターンで終了する。Ｂサブプログラム２０２は次にＣサブプログラム２０３をコールし、ただちにリターンで終了する。

20

以下同様である。最後に、コールされたＧサブプログラム２０８はＨサブプログラム２０９をコールし、ただちにリターンで終了する。一方Ｈサブプログラム２０９は単にリターンで終了する。

【００４１】Ｈアプリケーション２０９はコールしたＧサブプログラム２０８にリターンし、続いてただちにそれをコールしたサブプログラムにリターンし、以下同様にして連鎖を上昇する。リターン先のＣサブプログラム２０３はただちにＢサブプログラム２０２にリターンし、これはただちにＡサブプログラム２０１にリターンし、さらにこれはただちにメイン・サブプログラム２００にリターンする。

【００４２】上記で、サブプログラムの「終了」とは論理的終了である。すなわち、それはサブプログラムの実行中に実行される最後の文である。この「終了」は、物理的終了を意味しない。すなわち、この最後に実行される文がサブプログラムのリスティングの物理的に最後の文でなければならないことを意味する意図はない。

【００４３】従来のコンパイラ１１１は、ソース・プログラム１１０内で、図２のようなＣＡＬＬおよびＲＥＴＵＲＮ文に遭遇すると、図３～５に示される方法で応答する。これらの図は一般的なコンパイラの論理的機能を示す。この機能は、コンパイラの特定の実現例の機能に正確に一致しないこともある。

【００４４】図３を参照すれば、ステップ３００で、ＣＡＬＬ文に遭遇すると、ステップ３０１で、コンパイラ１１１は、コール側のサブプログラム（例えば、図２のメイン・サブプログラム２００）によって被コール側のサブプログラム（例えば、図２のＡサブプログラム２０１）に渡されたパラメータを計算するアプリケーション・オブジェクト・プログラム１１２命令を生成する。

【００４５】次にステップ３０２で、コンパイラ１１１は一般的に、このパラメータをスタック１１４の領域７０１にプッシュし、対応してスタック・ポインタ６０２をインクリメントする命令を生成する。しかし、あるプロセッサでは、サブプログラム間のパラメータの受け渡しはスタック１１４を通してではなく大域レジスタ（図示せず）を通して行われる。このようなプロセッサの場合、コンパイラ１１１は、図３の破線で示されるようにステップ３０２を省略する。さらに、実際は、ステップ３０１および３０２はパラメータごとに混合することができる。

【００４６】次に、コンパイラ１１１は、ステップ３０３で、フレーム・ポインタ６０１の値をスタック１１４の領域７０２にプッシュする命令を生成する。ステップ３０４で、フレーム・ポインタ６０１をスタック・ポインタ６０２と等置し、これによってフレーム・ポインタ６０１を次のフレームの開始点を指示するように移動する命令を生成する。ステップ３０５で、ステップ３０３を説明するためにスタック・ポインタ６０２をインクリ

メントする命令を生成する。次にコンパイラ１１１は、ステップ３０６で、実行されるべき次のオブジェクト・コード命令を指示する命令ポインタ６０３をスタック１１４の領域７０３にプッシュし、このスタック１１４エントリを説明するためにスタック・ポインタ６０２をインクリメントする命令を生成する。

【００４７】次に、あるコンパイラ１１１は、ステップ３０７で、コール側のサブプログラムのコンテクストをスタック１１４の領域７０４にプッシュし、このエントリを説明するためにスタック・ポインタ６０２をインクリメントする命令を生成する。コンテクストとは、例えば汎用レジスタ６０４の内容のように、コンピュータ１００がコール側のサブプログラムの実行を再開するために必要な情報である。しかし、ほとんどのコンパイラは、メモリ最適化として、ステップ３０７と同等のステップを異なるロケーションで実行する。従って、ステップ３０７は図３において破線で示されている。

【００４８】スタック１１４に情報を格納する命令の生成を完了すると、コンパイラ１１１は、ステップ３０８で、被コール側のサブプログラムに分岐する命令を生成する。

【００４９】この時点まで、アプリケーション・オブジェクト・プログラム命令のコンパイラ１１１による生成順序は実行順序に従っている。実行プログラム１１１になった場合のアプリケーション・オブジェクト・プログラムの実行順序は図３～５に点線で示されている。ステップ３０８で生成された分岐命令によって、アプリケーション・オブジェクト・プログラムの実行は図４の被コール側のサブプログラムとともに継続し、被コール側のサブプログラムおよびそれが呼び出した全サブプログラムの実行の終了後にのみコールしたサブプログラムにリターンする。

【００５０】リターン後、コールしたサブプログラムによってコールされたサブプログラムにステップ３０２で渡されたパラメータは不要となるため、コンパイラ１１１は次にステップ３０９でスタック・ポインタ６０２をデクリメントすることによって対応するパラメータをスタック１１４からポップする命令を生成する。ステップ３０２とともに説明されたように、あるプロセッサは、スタック１１４を通してパラメータを渡さず、ステップ３０２は除去される。ステップ３０２がない場合、ステップ３０９も同様に存在しない。従って、ステップ３０９は図３の破線によってバイパスされて示されている。

【００５１】この時点で、ＣＡＬＬ文によって開始された命令生成は終了し、コンパイラ１１１は、ステップ３１０で、アプリケーション・ソース・プログラム１１０の次の文の処理に進む。同様に、アプリケーション・オブジェクト・プログラムが実行プログラム１１１になったとき、オブジェクト・コードの実行は、アプリケーション・ソース・プログラムの次の文から生じたオブジェクト・コード命令に進む。

【００５２】図４を参照すると、ステップ４００で、アプリケーション・ソース・プログラム１１０内のサブプログラム（例えば被コール側のサブプログラム）の始め、すなわち開始文に遭遇すると、一般的なコンパイラは、ステップ４０１で、上記のステップ３０７と同等な動作を実行する。しかし、コンパイラが、図３のステップ３０７を実行した種類のものである場合、コンパイラは図４のステップ４０１を実行しない。このことは図４でステップ４０１をバイパスする破線によって示されている。

【００５３】次に、ステップ４０２で、コンパイラ１１１は、呼び出されたサブプログラムの局所および一時変数のためのスタック１１４上の領域７００を確保するため、スタック・ポインタ６０２をインクリメントする命令を生成する。こうして、ステップ４０３で、準備は終了し、コンパイラ１１１は、呼び出されたサブプログラムの本体の処理に進む。

【００５４】このように、アプリケーション・オブジェクト・プログラム１１１の実行は、図３のステップ３０８の分岐命令から、図４で生成された命令に進む。

【００５５】図５を参照すると、ステップ５０１で、被コール側のサブプログラムにおいてＲＥＴＵＲＮ文に遭遇すると、コンパイラ１１１は、ステップ５０２で、スタック・ポインタ６０２のをフレーム・ポインタ６０１の値と等しく設定する命令を生成する。これは、スタック１１４からリターン元のサブプログラムの全フレームをポップする効果を有する。

【００５６】次にステップ５０３で、コンパイラ１１１は、コール側のサブプログラムの実行の再開の準備をするため、スタック１１４から、リターン元のサブプログラムをコールしたサブプログラムのコンテクストにコンピュータ１００を復元する命令を生成する。図３のステップ３０３～３０７を参照すると、コール側のサブプログラムのコンテクストは、スタック１１４上で、現在スタック・ポインタ６０２によって指示されているポップされたフレームの始点の２ワード後に位置する。

【００５７】次に、ステップ５０４で、コンパイラ１１１は、スタック１１４から、格納されたフレーム・ポインタの値をフレーム・ポインタ・レジスタ６０１にロードする命令を生成する。図３のステップ３０３を参照すると、格納されたフレーム・ポインタの値は、現在スタック・ポインタ６０２によって指示されているポップされたフレームのワードにある。ステップ５０４で、フレーム・ポインタ６０１は、被コール側のサブプログラムのスタック・フレームの指示から、コール側のサブプログラムのスタック・フレームの指示にリターンする。

【００５８】また、ステップ５０５で、コンパイラ１１１は、スタック１１４から、格納された命令ポインタの値を命令ポインタ・レジスタ６０３にロードする命令を

生成する。図3のステップ303～306を参照すると、格納された命令ポインタの値は、現在スタック・ポインタ602によって指示されているワードの次の、ポップされたフレームのワードにある。この時点で、RETURN文によって開始された命令生成は終了し、ステップ506で、コンパイラ111はアプリケーション・ソース・プログラム110の次の文の処理に進む。

【0059】図5において点線で示されているように、アプリケーション・オブジェクト・コードの実行は、オブジェクト・プログラムが実行プログラム111になったとき、ステップ505まで図5の命令生成の順序に従う。ステップ505の命令の実行に応じて、アプリケーション・オブジェクト・プログラムの実行は、命令ポインタ603によって指示される命令に分岐し、これによってコール側のサブプログラム、すなわち、図3のステップ309で生成された命令にリターンする。

【0060】図2に示されたCALLおよびRETURN文の連鎖に対するコンパイラ111の応答の最終結果は、実行プログラム111になった後に、図6から10に示された方法でメモリ101のスタック114を操作するアプリケーション・オブジェクト・プログラムである。メイン・サブプログラム200の実行中およびAサブプログラム201のコールの前には、スタック114は、図6に示されるように、メイン・サブプログラム200の局所および一時変数の領域700を含むメイン・サブプログラム200のフレーム610からなる。

【0061】Aサブプログラム201へのコールによって、フレーム610は、コールされたAサブプログラム201のパラメータの領域701を追加することによって拡張され、呼び出されたAサブプログラム201のための新フレーム611も作成される。新フレーム611は、図7に示されるように、先行フレーム610へのフレーム・ポインタを含む領域702と、コール側のメイン・サブプログラム200の命令内のリターン地点への命令ポインタを含む領域703と、コール側のメイン・サブプログラム200のコンテクストを含む領域704と、コールされたAサブプログラムの局所および一時変数のための領域700を有する。

【0062】同様に、図8に示されるように、Bサブプログラム202へのコールによって、コール側のAサブプログラム201のフレーム611が拡張され、コールされたBサブプログラムのための新フレーム612も作成される。この処理は継続し、図9に示されるように、Gサブプログラム208へのコールの時点で、スタック114は先行する各サブプログラムのフレームを含み、Gサブプログラム208へのコールによって、Gサブプログラム208のためのフレーム618がスタック114に追加される。最後に、図10に示されるように、Hサブプログラム209へのコールによってフレーム618は拡張され、Hサブプログラム209のためのフレーム619がスタック114に追加される。

【0063】Hサブプログラム209からGサブプログラム208へのリターンによってHサブプログラム209のフレーム619がスタック114から削除され、スタック114が図9に示される状態に復元される。対応する結果が、図2の連鎖の連続する各サブプログラムからのリターンに対して得られ、最後に、Cサブプログラム203からBサブプログラム202へのリターンによって、スタック114は図8に示された状態に復元され、Bサブプログラム202からAサブプログラム201へのリターンによってスタック114は図7に示される状態に復元され、Aサブプログラム201からメイン・サブプログラム200へのリターンによってスタック114は図6に示される状態に復元される。

【0064】上記の従来の動作と対照して、本発明によるコンピュータ100の動作は図11～20に示されるようになり、以下で説明される。

【0065】アプリケーション・ソース・プログラム110において、図2のサブルーチン終端CALLおよびRETURN文の連鎖は、図11に示されるPASS_CONTROL文の連鎖に置き換えられる。メイン・サブプログラム200は、この場合も、Aサブプログラム201へのコールによって連鎖を開始する。しかし、Aサブプログラム201は、Bサブプログラム202へパス・コントロールすることによって機能を終了し、Bサブプログラム202はCサブプログラム203へパス・コントロールすることによって機能を終了し、などとなる。最後に、Gサブプログラム208はHサブプログラム209へパス・コントロールすることによって機能を終了し、Hサブプログラム209は単にリターンによって終了する。

【0066】しかし、図1の従来の例とは異なり、Hサブプログラム209からのリターンはA201からG208までのサブプログラムの全連鎖をバイパスして、連鎖を開始したメイン・サブプログラム200に直接リターンする。Hサブプログラム209の実行が終了すると、中間でA201からG208までの他のサブプログラムの実行を再開することなく、メイン・サブプログラム200の実行はただちに再開される。従って、コール側から被コール側のサブプログラムへの一段ごとのリターンは削減されて単一のリターンにまとめられることにより、個々の一段ごとのリターンによって消費される処理時間が節約される。

【0067】上記のように、A201からG208までの呼び出された各サブプログラムは、次に呼び出されるファンクションへパス・コントロールすることによって機能を終了する。これは、サブプログラムは暗黙に終了することも、明示的に終了することもあるためである。明示的には、サブプログラムの最後に実行される文が新たなPASS_CONTROL文であって、これは、呼

び出されたサブプログラムへパス・コントロールし、呼び出し側のサブプログラムの終端を規定するために使用される。

【００６８】暗黙には、Ａ２０１からＧ２０８までのサブプログラムは、それぞれ、新たなＰＡＳＳ＿ＣＯＮＴＲＯＬ文の後に、従来のＲＥＴＵＲＮ文で終了することがある。しかし、リターンはHサブプログラム２０９からメイン・サブプログラム２００へ直接行われるため、Ａ２０１からＧ２０８までのサブプログラムをＲＥＴＵＲＮ文で終了することは、たとえそれが可能であっても、不要であり余分である。Ａ２０１からＧ２０８までのサブプログラムにおいてＰＡＳＳ＿ＣＯＮＴＲＯＬの後にＲＥＴＵＲＮ文が含まれる場合、ＲＥＴＵＲＮ文に応じてコンパイルされたアプリケーション・オブジェクト・コード命令はバイパスされ、実行されることはない。

【００６９】さらに、暗黙には、Ａ２０１からＧ２０８までのサブプログラムは、従来のＣＡＬＬおよびＲＥＴＵＲＮ文で従来のように終了することも可能である。言い換えれば、アプリケーション・ソース・プログラムにおいて、図２のＣＡＬＬおよびＲＥＴＵＲＮ文の連鎖の置換は、例えばアプリケーション・プログラム１１０を書くときにプログラマによって明示的に行われる必要はなく、コンパイラ１１１自体によってアプリケーション・プログラム１１０のコンパイル中に暗黙に行われることが可能である。この例では、コンパイラ１１１は、サブプログラムの終端の文を検査して、ＣＡＬＬおよびＲＥＴＵＲＮ文が実行されるサブプログラムの最後の２文であるかどうかを判定し、そうである場合には、それらをＰＡＳＳ＿ＣＯＮＴＲＯＬ文と等価なオブジェクト・コード命令に変換する。

【００７０】呼び出されたサブプログラムの連鎖の終端からその連鎖の開始点へ直接単一のリターンを実行する上記の機能は、Ａ２０１からＨ２０９までの呼び出されたサブプログラムによるスタック・フレームの共有によって容易にされ、実現される。言い換えれば、呼び出された各サブプログラムのための新たなスタック・フレームは作成されず、すべての呼び出されたサブプログラムに対して１個のスタック・フレームのみが作成され、順に使用される。

【００７１】本発明によるコンパイラ１１１は、図１２～１４に示される方法で、ソース・プログラム１１０におけるＰＡＳＳ＿ＣＯＮＴＲＯＬおよびＲＥＴＵＲＮ文との明示的または暗黙の遭遇に対応する。コンパイラ１１１は、暗黙のＰＡＳＳ＿ＣＯＮＴＲＯＬ文でない従来のＣＡＬＬおよびＲＥＴＵＲＮ文に、上記の従来の方法で対応する。図３～５の場合のように、図１２～１４はコンパイラの論理的機能を示す。

【００７２】図１２を参照すれば、ステップ１２００でＣＡＬＬ文に遭遇すると、コンパイラ１１１は、ステップ１２０１でアプリケーション・ソース・プログラム１１０の次の文をチェックし、ステップ１２０２でそれがＲＥＴＵＲＮ文であるかどうか判定することによって、前記ＣＡＬＬ文が暗黙のＰＡＳＳ＿ＣＯＮＴＲＯＬ文であるかどうかチェックする。次の文がＲＥＴＵＲＮ文でない場合、前記ＣＡＬＬ文は暗黙のＰＡＳＳ＿ＣＯＮＴＲＯＬ文ではなく、コンパイラ１１１は図３に示された方法でそのＣＡＬＬ文を従来のように処理する。しかし、ＣＡＬＬ文に続く文がＲＥＴＵＲＮ文である場合、そのＣＡＬＬ文は暗黙のＰＡＳＳ＿ＣＯＮＴＲＯＬ文であり、コンパイラ１１１はステップ１２１１に進む。

【００７３】また、ステップ１２１０で、明示的ＰＡＳＳ＿ＣＯＮＴＲＯＬ文に遭遇すると、コンパイラ１１１は直接ステップ１２１１に進む。

【００７４】ステップ１２１１で、コンパイラ１１１は、呼び出されたサブプログラムへ呼び出し側のサブプログラムによって渡されているパラメータを計算するアプリケーション・オブジェクト・プログラム１１２の命令を生成する。これは、図３のステップ３０１と同等である。次にコンパイラ１１１は、一般的に、パラメータをスタック１１４にプッシュする命令を生成する。

【００７５】サブプログラムは、スタック上のそれ自身のフレームに先行するフレームにパラメータを発見することを予期する。しかし、この例では、呼び出されたサブプログラムに対して新たなスタック・フレームは作成されない。従って、パラメータは、先行するフレーム、すなわち、メイン・サブプログラム２００のフレーム６１０のパラメータ領域に格納されなければならない。これを行うため、コンパイラ１１１は次のファンクションを実行する。

【００７６】コンパイラ１１１は、ステップ１２１２で、スタック・ポインタ６０２の現在の値を一時的にセーブする命令を生成し、ステップ１２１３で、スタック・ポインタ６０２を、呼び出されたサブルーチンに渡されているパラメータの数をフレーム・ポインタ６０１から引いた値に設定する命令を生成する。これは、スタック・ポインタがメイン・サブルーチン２００のスタック・フレーム６１０のパラメータ領域７０１を指示する効果を有する。

【００７７】スタック・ポインタ６０２がパラメータ領域７０１の外側、すなわち、前方の場所を指示しないことを保証するため、パラメータの最大許容数に制限が課され、パラメータの許容数のための空間が図１２のステップ１２１３においてスタック上で使用可能であることを保証するために、図３のステップ３０２で前もって生成された命令がその最大許容数だけスタック・ポインタ６０２をインクリメントしなければならない。これは、Ａサブプログラム２０１を十分多くのダミー・パラメータとともにコールしてパラメータの総数が最大数に等しくなるようにするか、または、パラメータ領域として確

27

保される大きさを宣言することが可能な「ｐｒａｇｍａｔ」文をサポートすることによって可能となる。

【００７８】スタック・ポインタ６０２が先行スタック・フレーム６１０のパラメータ領域７０１を指示すると、コンパイラ１１１は、ステップ１２１４で、計算されたパラメータをスタック１１４上にプッシュし、対応してスタック・ポインタ６０２をインクリメントする命令を生成する。これは、図３のステップ３０２と同様である。次にステップ１２１５で、コンパイラ１１１は、ステップ１２１２でセーブされた値をスタック・ポインタ６０２に復元し、それによってスタック・ポインタ６０２が再びスタック１１４の「トップ」を指示するようにする命令を生成する。

【００７９】上記では、コンパイラ１１１は一般的にパラメータをスタック上にプッシュする命令を生成するとした。しかし、図３とともに説明されたように、あるプロセッサでは、パラメータはスタック１１４を通してではなく大域レジスタを通して渡される。この場合、コンパイラ１１１は、図１２の破線で示されるように、ステップ１２１２～１２１５を省略する。

【００８０】次に、ステップ１２１６で、コンパイラ１１１は、呼び出されたサブプログラムへ分岐する命令を生成する。以下で明らかになるように、呼び出されたサブプログラムはいくつかのエントリ・ポイントを有する。図３のステップ３０８の場合のように呼び出されたサブプログラムの開始点に分岐する命令を生成する代わりに、コンパイラ１１１は、呼び出されたサブプログラムにおいて、呼び出されたサブプログラムの開始点からある所定量（実現オプションに依存する）をオフセットとするポイントに分岐する命令を生成する。

【００８１】この時点で、ＰＡＳＳ＿ＣＯＮＴＲＯＬ文によって開始された命令生成は終了し、ステップ１２１７で、コンパイラ１１１はアプリケーション・ソース・プログラム１１０の次の文の処理に進む。この処理は従来のものでよい。

【００８２】また、図１２の破線および破線ボックスで示されるように、コンパイラ１１１は、ステップ１２１８で、アプリケーション・ソース・プログラムにおける次の文をチェックして、それがＲＥＴＵＲＮ文であるかどうか判定することが可能である。上で説明されたように、明示的または暗黙のＰＡＳＳ＿ＣＯＮＴＲＯＬ文に続くＲＥＴＵＲＮ文は余分であり、コンパイラ１１１は、それを無視して、ステップ１２１７に戻り、そのＲＥＴＵＲＮ文に続く次のアプリケーション・ソース・コード文に進み、処理することが可能である。

【００８３】しかし、ステップ１２１８で、暗黙または明示的ＰＡＳＳ＿ＣＯＮＴＲＯＬ文に続く文がＲＥＴＵＲＮ文でない場合、コンパイラ１１１は、ステップ１２１９で、従来のコンパイルを継続する。ＰＡＳＳ＿ＣＯＮＴＲＯＬ文に続く文への到達可能性の判定基準はＲＥＴＵＲＮ文に対するものと同一である。

28

【００８４】ステップ１２１６まで、コンパイラ１１１によるアプリケーション・オブジェクト・プログラム命令１１２の生成順序は、実行順序に従う。実行プログラム１１１になったときのアプリケーション・オブジェクト・プログラムの実行順序は図１２～１４の点線で示される。ステップ１２１６で生成される分岐命令のため、アプリケーション・オブジェクト・プログラムの実行は図１３の被コール側のサブプログラムに継続する。

【００８５】図１３は、生成されるアプリケーション・オブジェクト・プログラムが複数のエントリ・ポイントを有することを除いては、図４と機能的に等価である。従って、ステップ１３００で、アプリケーション・ソース・プログラム１１０においてサブプログラム（例えば呼び出されたサブプログラム）の開始すなわちエントリ文に遭遇すると、コンパイラ１１１は、図４で説明されたように図４と同等のファンクションを実行する。

【００８６】図１３においてコンパイラ１１１によって生成されたアプリケーション・オブジェクト・サブプログラムの実行は図４で説明されたように進行する。すなわち、従来のＣＡＬＬ文の結果として従来のようにこのサブプログラムへ分岐すると、図３でステップ３０７が実行されていない場合、オブジェクト・サブプログラムの実行はステップ１３０１で生成されたコードで開始し、図３でステップ３０７が実行された場合は、ステップ１３０２で生成されたコードで開始する。これは、図４で説明されたことと直接対応する。

【００８７】暗黙または明示的ＰＡＳＳ＿ＣＯＮＴＲＯＬ文の結果として図１３のオブジェクト・サブプログラムの実行に分岐した場合、呼び出し側のコンテクストをセーブする必要はない。従って、オブジェクト・サブプログラムの実行は、ステップ１３０１で生成された命令の後のポイント（もし存在すれば）から開始する。

【００８８】さらに、スタック・フレームの共有に関する２つのオプションが本実施例において提供される。第１のオプションは、呼び出されたサブプログラムが共有スタック・フレーム１６２０の全領域７００～７０４を共有することである。第２のオプションは、呼び出された各サブプログラムがそれ自身の局所および一時変数領域７００を共有スタック・フレーム１６２０内に有することである。

【００８９】第１オプションのもとで、コンパイラ１１１は、図１２のステップ１２１６で、アプリケーション・オブジェクト・サブプログラムにおいてオブジェクト・サブプログラムの開始点からの固定オフセットであるポイントへ分岐する命令を生成し、ステップ１３０３で生成された命令で開始することにより、図１３で生成される全命令を実効的にバイパスする。

【００９０】第２オプションのもとで、コンパイラ１１１は、再び図１２のステップ１２１６で、アプリケーシ

29

ョン・オブジェクト・サブプログラムにおいて図１３のステップ１３０２で生成された命令で開始する固定オフセット・ポイントへ分岐する命令を生成する。これらのオフセット・ポイントは、コンパイラによって処理された全サブプログラムに対して固定されている、すなわち、同一である。

【００９１】図１４を参照すれば、ステップ１４００で、呼び出されたサブプログラムにおいて、図１２のステップ１２１８～１２１９に示される方法でオプションとして処理されないＲＥＴＵＲＮ文に遭遇すると、コンパイラは、図５に示されるファンクションを実行することによって従来の方法で対応する。

【００９２】アプリケーション・オブジェクト・プログラムが実行プログラム１１１になった場合、アプリケーション・オブジェクト・コードの実行中には、Ｈサブプログラム２０９のＲＥＴＵＲＮ文に応じてコンパイラによって生成されたコードは、図１１のＡ２０１からＨ２０９までのサブプログラムをコンパイルした結果として生成されたアプリケーション・オブジェクト・コードにおいて遭遇する最初で唯一のＲＥＴＵＲＮコードである。

【００９３】Ｈサブプログラム２０９に対して図５のステップ５０２でコンパイラによって生成されたコードの実行により、メイン・サブプログラム２００のフレーム６１０に続く全情報がスタック１１４からポップされる。ステップ５０３～５０５でコンパイラによって生成されたコードの実行により、コンピュータ１００は、メイン・サブプログラム２００がＡサブルーチン２０１を呼びだしたときの状態に復元される。従って、ステップ５０５で実行される分岐に応じて、実行は、メイン・サブプログラム２００の、図３のステップ３０９で生成される命令に復帰する。この過程で、Ａ２０１からＧ２０８までのサブプログラムのＲＥＴＵＲＮ文に対して生成された可能性のあるオブジェクト・コードはスキップされ、実行されることはない。

【００９４】図１１に示される暗黙または明示的ＰＡＳＳ＿ＣＯＮＴＲＯＬ文の連鎖に対するコンパイラ１１１の応答の最終結果は、実行プログラム１１１になった場合、図１５～２０に示される方法でメモリ１０１内のスタック１１４を操作するアプリケーション・オブジェクト・プログラム１１２である。図１５および１６に示されるように、メイン・サブプログラム２００によるＡサブプログラム２０１へのコールの前後で、スタック１１４は、従来技術に対して説明されそれぞれ図６および７で示されたものと同一である。しかし、図１６で、Ａサブプログラム２０１のスタック・フレームは以下の新たなファンクションを示すために「共有フレーム１６２０」と表示されている。

【００９５】図１７に示されるように、Ａサブプログラム２０１がＢサブプログラム２０２へパス・コントロー

30

ルした後、メイン・サブプログラム２００のフレーム６１０のパラメータ領域７０１は被コール側のＡサブプログラム２０１のパラメータを格納しておらず、呼び出されたＢサブプログラム２０２のパラメータを格納している。

【００９６】図１３のステップ１３０２とともに説明されたオプション１によれば、共有フレーム１６２０の局所および一時変数領域７００もまた被コール側のＡサブプログラム２０１の変数を格納しておらず、呼び出されたＢサブプログラム２０２の変数を格納している。しかし、オプション２によれば、共有フレーム１６２０の最初の局所および一時変数領域７００は被コール側のＡサブプログラム２０１の変数を格納し続け、共有フレーム１６２０は、呼び出されたＢサブプログラム２０２の変数を格納するための追加の局所および一時変数領域７００によって拡張される。

【００９７】スタック１１４に対し、遭遇するすべてのＰＡＳＳ＿ＣＯＮＴＲＯＬ文について、対応する変更が行われる。従って、Ｇサブプログラム２０８の呼び出し後、図１８に示されるように、フレーム６１０のパラメータ領域７０１はＧサブプログラム２０８のパラメータを格納し、共有フレーム１６２０の単一の局所および一時変数領域７００はＧサブプログラム２０８の変数を格納しているか、または、共有フレーム１６２０はＡ２０１からＧ２０８までの各サブプログラムに対して１個ずつ存在する複数の領域７００を含むように拡張されている。

【００９８】図１９に示されるように、Ｈサブプログラム２０９の呼び出し後、フレーム６１０のパラメータ領域７０１はＨサブプログラム２０９のパラメータを格納し、共有フレーム１６２０の単一の局所および一時変数領域７００はＨサブプログラム２０９の変数を格納しているか、または、共有フレーム１６２０はＡ２０１からＨ２０９までの各サブプログラムに対して１個ずつ存在する複数の領域７００を含むように拡張されている。

【００９９】図２０に示されるように、Ｈサブプログラム２０９からのリターンに従って、スタック１１４は直接、ただちに、Ａサブプログラム２０１へのコールに先行する図１５に示される状態に復元される。

【０１００】もちろん、本発明の上記の実施例に対するさまざまな変更および修正が可能である。例えば、パラメータ領域は、再利用または共有を可能にするのに十分な大きさである限り、スタック上のどこに位置するかは重要でない。同じことが局所および一時変数領域についても成り立つ。

【０１０１】

【発明の効果】以上述べたごとく、本発明によれば、ＰＡＳＳ＿ＣＯＮＴＲＯＬと呼ばれる装置によって、相異なるサブプログラムの呼び出しの全系列からの、その系列を開始したサブプログラムへの直接の復帰が、その系

31

列内の中間呼び出しを実行したサブプログラムへの復帰を妨げることなく実行される。

【図面の簡単な説明】

【図１】本発明の実施例の環境として使用されるコンピュータの機能ブロック図である。

【図２】図１のコンピュータの従来のＣＡＬＬ－ＲＥＴＵＲＮ呼び出し系列の機能図である。

【図３】図２の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

【図４】図２の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

【図５】図２の文を含むプログラムをコンパイルするときの従来のコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

【図６】図１のコンピュータによって、図３～５のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図７】図１のコンピュータによって、図３～５のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図８】図１のコンピュータによって、図３～５のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図９】図１のコンピュータによって、図３～５のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図１０】図１のコンピュータによって、図３～５のコンパイルされた命令により実行される従来の実行スタック操作のブロック図である。

【図１１】図１のコンピュータのＰＡＳＳ＿ＣＯＮＴＲＯＬ－ＲＥＴＵＲＮ呼び出し系列の機能図である。

【図１２】図１１の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

32

【図１３】図１１の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

【図１４】図１１の文を含むプログラムをコンパイルするときのコンパイラの動作、および、コンパイルされた命令の図１のコンピュータによる実行のフロー図である。

【図１５】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図１６】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図１７】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図１８】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

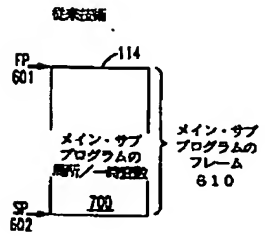
【図１９】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

【図２０】図１のコンピュータによって、図１２～１４のコンパイルされた命令により実行される実行スタック操作のブロック図である。

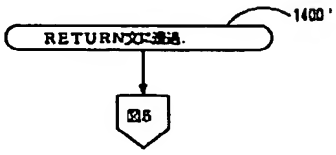
【符号の説明】

１００　コンピュータ
１０１　メモリ
１０２　ＣＰＵ
１１０　入力情報
１１１　実行プログラム
１１２　出力情報
１１４　実行スタック
６０１　フレーム・ポインタ・レジスタ
６０２　スタック・ポインタ・レジスタ
６０３　命令ポインタ・レジスタ
６０４　汎用レジスタ
６０５　レジスタ・ファイル
６０６　ＡＬＵ

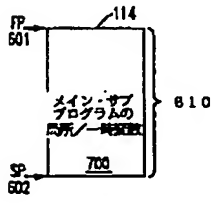
【図６】



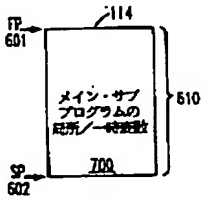
【図１４】



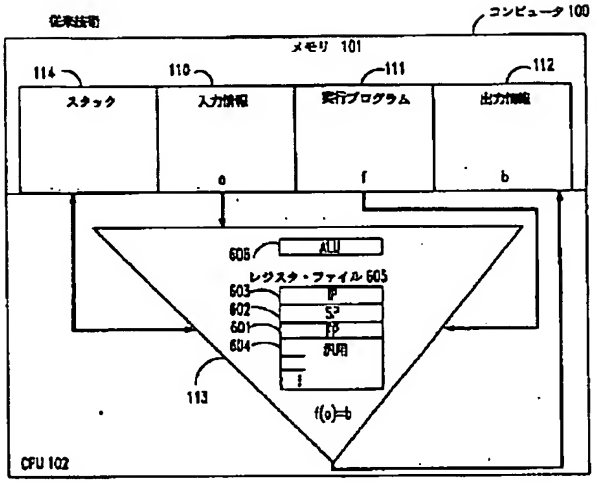
【図１５】



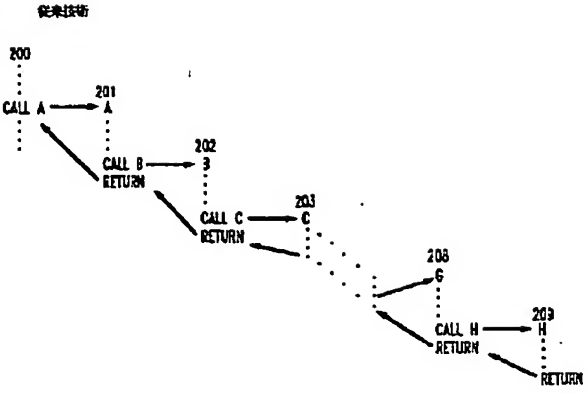
【図２０】



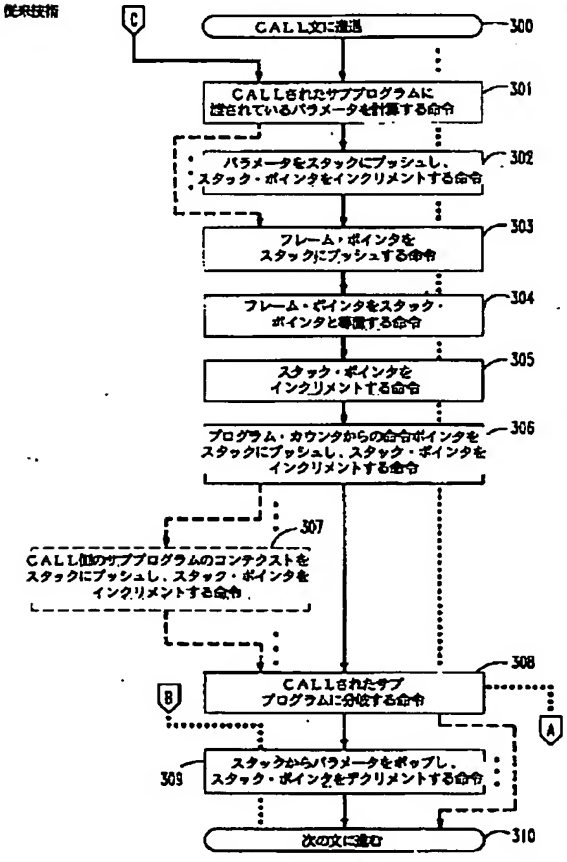
【図1】


従来技術
コンピュータ 100
メモリ 101
114
スタック
110
入力情報
111
実行プログラム
112
出力情報
a
f
b
605
ALU
レジスタ・ファイル 605
603
602
601
604
汎用
113
f(a)=b
CPU 102

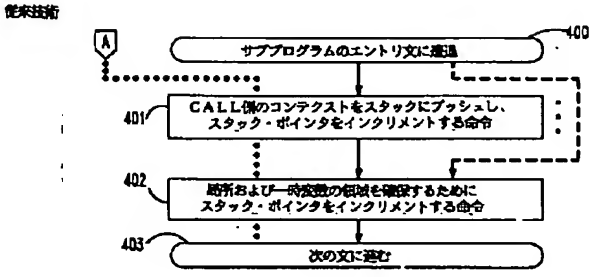
【図2】


従来技術
200
CALL A
201
A
CALL B
RETURN
202
B
CALL C
RETURN
203
C
208
G
CALL H
RETURN
209
H
RETURN

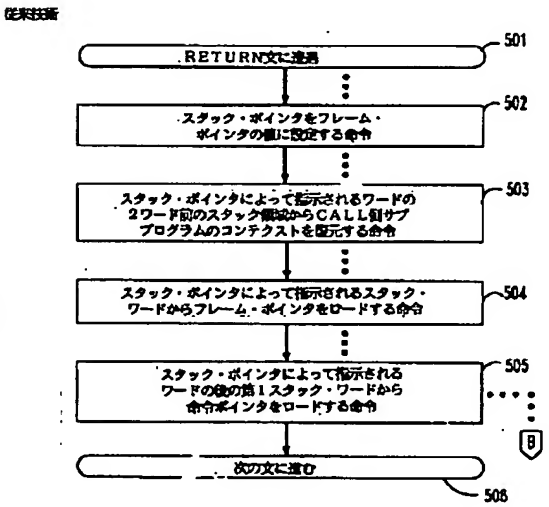
【図3】


従来技術
C
CALL文に遷移
300
CALLされたサブプログラムに
渡されているパラメータを計算する命令
301
パラメータをスタックにプッシュし、
スタック・ポインタをインクリメントする命令
302
フレーム・ポインタを
スタックにプッシュする命令
303
フレーム・ポインタをスタック・
ポインタと等値する命令
304
スタック・ポインタを
インクリメントする命令
305
プログラム・カウンタからの命令ポインタを
スタックにプッシュし、スタック・ポインタを
インクリメントする命令
306
CALL側のサブプログラムのコンテクストを
スタックにプッシュし、スタック・ポインタを
インクリメントする命令
307
B
CALLされたサブ
プログラムに分岐する命令
308
A
スタックからパラメータをポップし、
スタック・ポインタをデクリメントする命令
309
次の文に進む
310

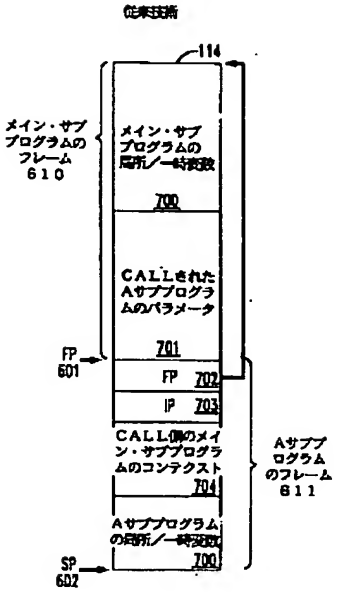
【図4】


従来技術
A
サブプログラムのエントリ文に遷移
400
CALL側のコンテクストをスタックにプッシュし、
スタック・ポインタをインクリメントする命令
401
局所および一時変数の領域を確保するために
スタック・ポインタをインクリメントする命令
402
次の文に進む
403

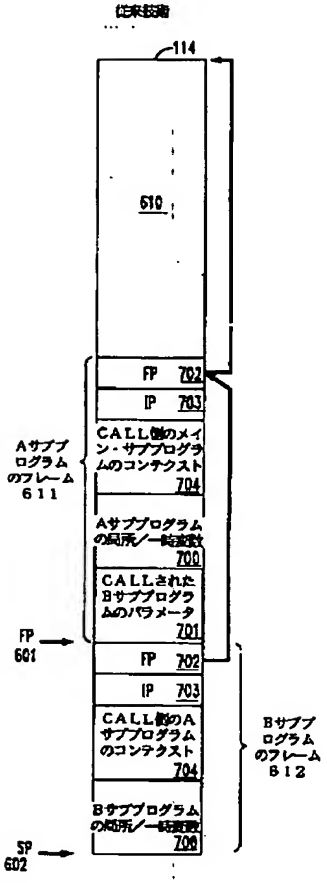
【図5】


従来技術
RETURN文に遷移
501
スタック・ポインタをフレーム・
ポインタの値に設定する命令
502
スタック・ポインタによって指示されるワードの
2ワード前のスタック領域からCALL側サブ
プログラムのコンテクストを復元する命令
503
スタック・ポインタによって指示されるスタック・
ワードからフレーム・ポインタをロードする命令
504
スタック・ポインタによって指示される
ワードの後の第1スタック・ワードから
命令ポインタをロードする命令
505
B
次の文に進む
506

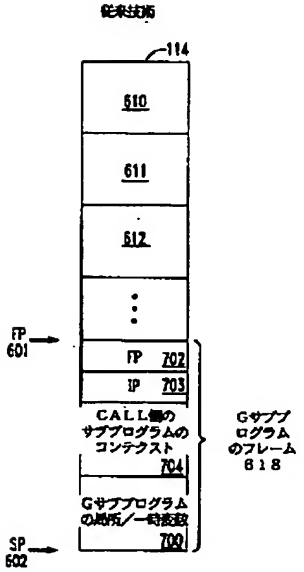
【図7】



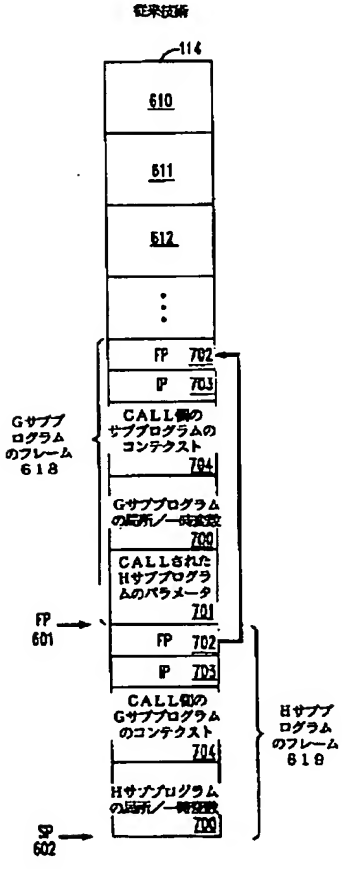
【図8】



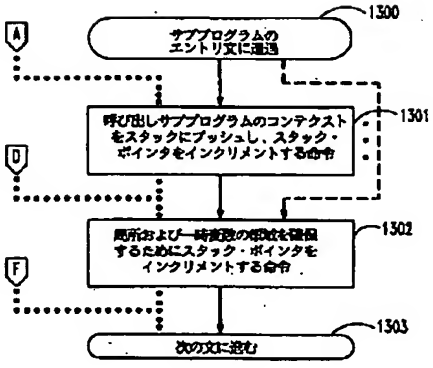
【図9】



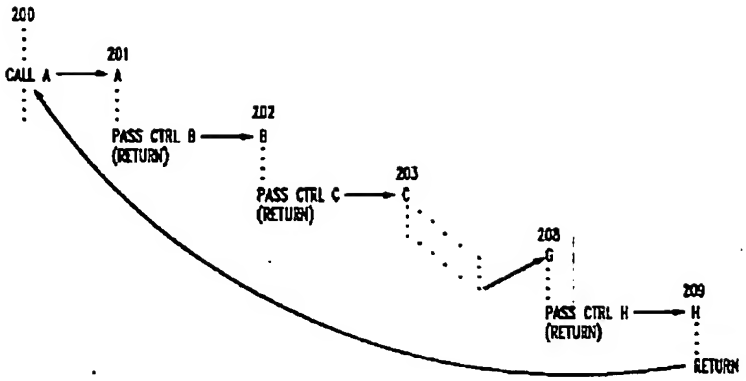
【図10】



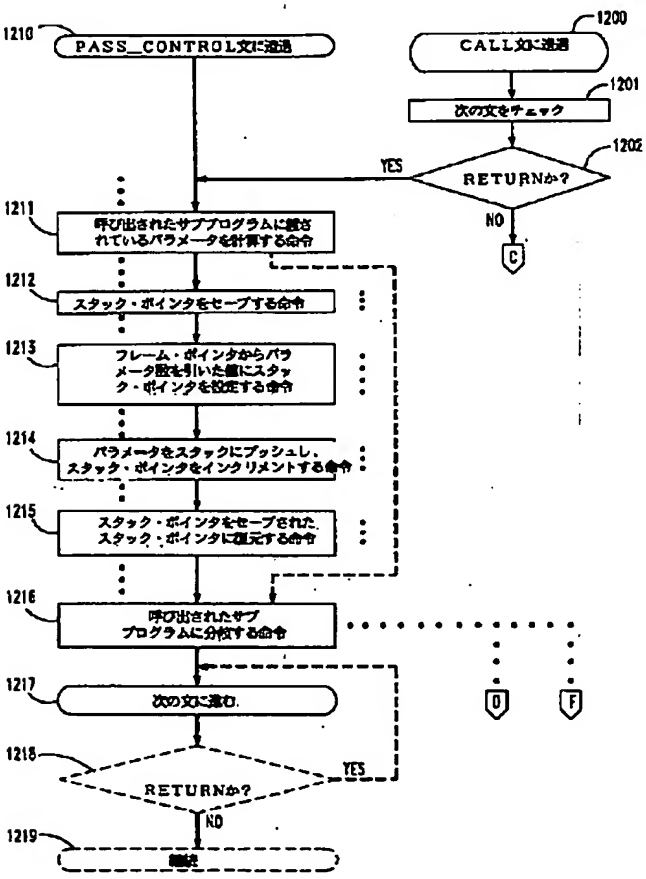
【図13】



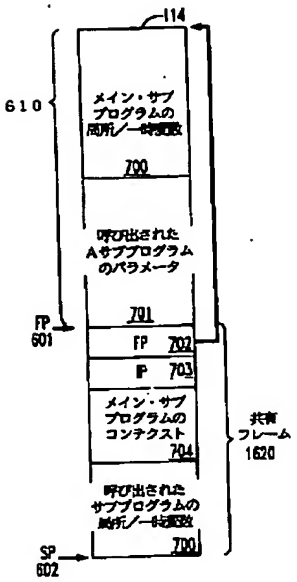
【図１１】



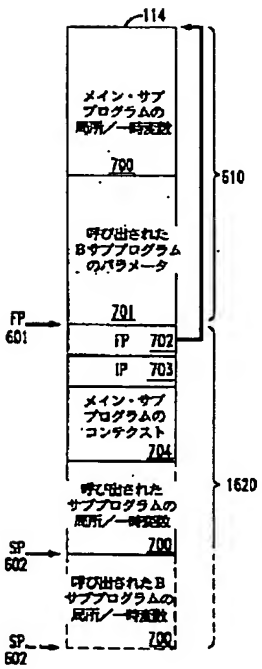
【図１２】



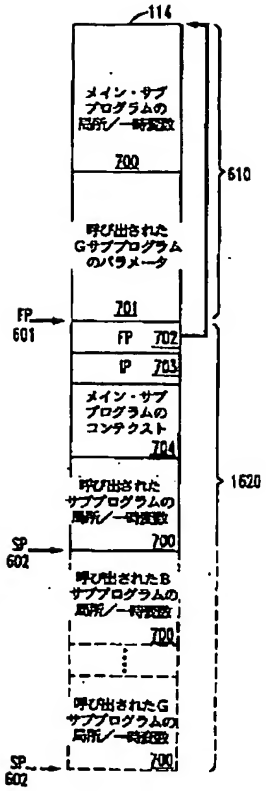
【図１６】



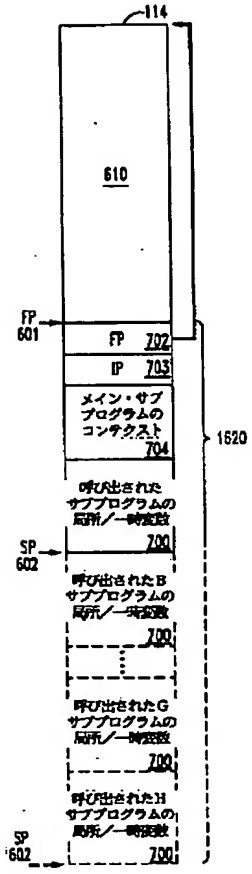
【図１７】



【図１８】


114
メイン・サブプログラムの局所／一時変数
700
610
呼び出されたGサブプログラムのパラメータ
701
FP
601
FP 702
IP 703
メイン・サブプログラムのコンテクスト
704
1620
呼び出されたサブプログラムの局所／一時変数
700
SP
602
呼び出されたBサブプログラムの局所／一時変数
700
呼び出されたGサブプログラムの局所／一時変数
700
SP
602

【図１９】


114
610
FP
601
FP 702
IP 703
メイン・サブプログラムのコンテクスト
704
1620
呼び出されたサブプログラムの局所／一時変数
700
SP
602
呼び出されたBサブプログラムの局所／一時変数
700
呼び出されたGサブプログラムの局所／一時変数
700
呼び出されたHサブプログラムの局所／一時変数
700
SP
602